Win Forms - Advanced Creating Controls

Shawn Burke
Microsoft Corporation

1-421


Microsoft®
PDC 2000
Professional Developers Conference
Microsoft®
.net™
the defining point

Introduction

- Types of Controls
- Painting
- Properties, Events and Metadata
- Customizing design time support
- Licensing

The Basics

- Two types of controls:
  - “Custom” controls
  - “User” or composite controls
- Integrated designer support
  - .NET Framework built with design-time in mind
  - Design-time built using .NET Framework
    - VS .NET Property Browser, Server Explorer, Win Forms

Controls

- What to derive from and when:
  - Control
  - Rich Control
  - Container Control
  - Scrollable Control
  - User Control
  - Form

Painting

- GDI+:

```
protected override void OnPaint(PaintEventArgs pe) {
   Graphics g = pe.Graphics;
   Brush backgroundBrush =
      new TextureBrush(new Bitmap("colorbars.jpg"));
   Brush washBrush =
      new SolidBrush(Color.FromARGB(180, Color.White));
   g.FillRectangle(backgroundBrush, ClientRectangle);
   g.FillRectangle(washBrush, ClientRectangle);
   g.DrawString(“Hello”, Font, Brushes.Green, 15, 15);
}
```

Painting

- Opacity
- Non-rectangular windows

DEMO!

Properties

- Properties are “smart fields”
  - Natural syntax and accessors

```
public class Button: Control {

   private string text;

   public string Text {
      get {
         return text ;
      }
      set {
         text = value;
         Repaint();
      }
   }
}
```

```
Dim b As New Button
b.Text = "OK"
Dim s As String = b.Text
```

Events

- Two Event classes:
  - The EventHandler delegate class
    - Hooks up event to event handlers
  - The EventArgs class
    - Contains information about the event that
      has been fired
- Three methods on a component:
  - AddOn<EventName>
  - RemoveOn<EventName>
  - On<EventName>

Events

```
public class Button: Control
{
   private EventHandler clickHandler;

   public void AddOnClick(EventHandler handler) {
      clickHandler =
         (EventHandler)Delegate.Combine (clickHandler , handler);
   }
```

```
      ....
      //Register the event handler
      button1.AddOnClick(AddressOf buttonClicked)
      ....
      //The event handling method
      private Sub buttonClicked(sender As Object, ev As EventArgs)
          MessageBox.Show("Hello Win Forms World!")
      End Sub
      ....
```

```
}
```

Metadata And Attributes

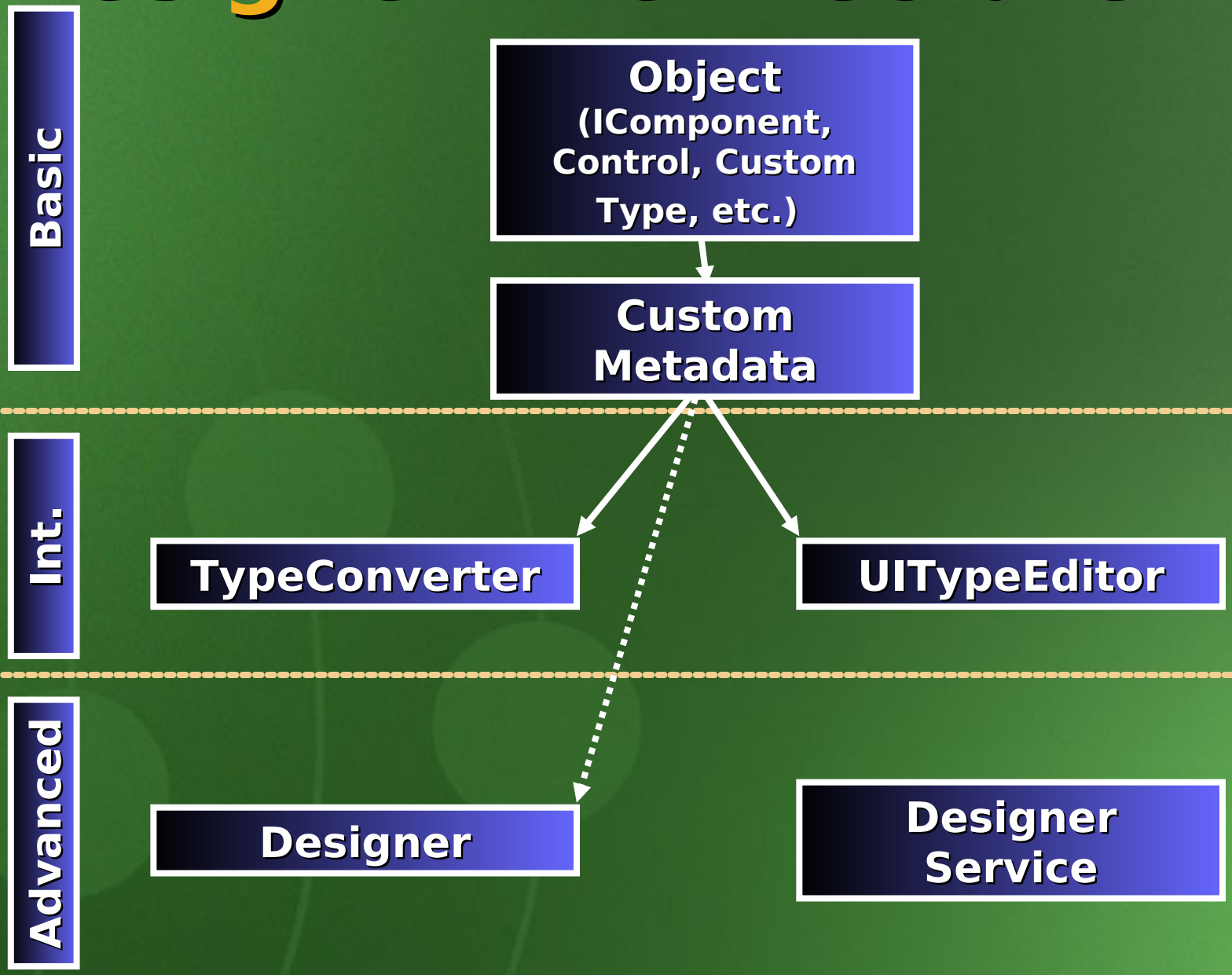
- How do you define additional information about a class?

```
[
    Category("Appearance"),
    Description(“Text to display"),
    DefaultValue(“OK”),
    Bindable(true)
]
public string Text {
        ....
    }
}
```

Metadata And Attributes

```
Overrides Public Property <Browsable(False)> ForeColor As Color
    Get
        return MyBase.ForeColor
    End Get
    Set
        'No Action
    End Set
End Property
```

Designer Architecture



Type Converters

- Runtime and design-time
- No dependency on any UI package
- Base type:
  - System.ComponentModel.TypeConverter
- Enable:
  - Type conversion (ConvertFrom, ConvertTo)
  - Persistence (CreateInstance, PersistInfo)
  - Custom properties
  - Standard values
  - Validity checking

Property Editors

- Editing UI for hosts
  - Property Browsers, DataGrids
- EditorAttribute enables multiple editors
  - Extensible, however today we only have one type
- System.Drawing.Design.UITypeEditor
  - Used by the property browser

UITypeEditor

- PaintValue
  - Enables custom rendering for each value
  - Example: Color Property
- EditValue
  - Enables Custom Editing UI
  - Example: Color Picker

DEMO!

Licensing Goals

- 100% functionality of COM licensing
- Simplify licensing model
- Support server component licensing
- Component architecture for license providers

Licensed Win Forms Control

```
using System;
using System.WinForms;
using System.ComponentModel;

[LicenseProvider(typeof(LicFileLicenseProvider))]
public class MyControl : RichControl {
        private License license;
        public MyControl() {
            license =
              LicenseManager.Validate(typeof(MyControl), this);
        }

        protected override void Dispose() {
                if (license != null) {
                    license.Dispose();
                    license = null;
                }
        }
}
```

Design Points

- LicenseProvider abstraction
  - Provides reuse of licensing validation and granting logic
  - Allows licensed components to be implemented independent of licensing logic
- Metadata bindings component to provider through custom attribute

LicenseProvider

```
public abstract class LicenseProvider {
    public abstract License GetLicense(
                        LicenseContext context,
                        Type type,
                        object instance,
                        bool allowExceptions);
}
```

- Context - Information about the license request
- Type - Type requesting license
- Instance - Instance of type actually being created
- AllowExceptions - If true, then GetLicense may throw an exception

Summary

- Easy to build controls
- Easy to get cool painting effects
- Easy to integrate into design-time
- Flexible licensing architecture

Related Sessions And References

- Other PDC sessions recommended:
  - Security Considerations for Download Controls (1-222)
  - GDI+ and Win Forms (1-212)
  - Deploying and Versioning Your Application (3-200)
  - .NET Framework Security (3-300)
- Win Forms Quick Start in the SDK
- Whitepaper – *"Creating Designable Components for the Visual Studio 7*

Questions?

Where do you want to go today?

Microsoft